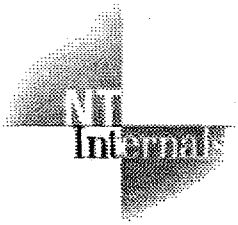

Inside NTFS

NT's native file system—past, present, and future

by Mark Russinovich

When Microsoft first released Windows NT, it came with support for three file systems: FAT, HPFS, and NTFS. Microsoft originally designed FAT for DOS, IBM introduced HPFS for OS/2, and Microsoft created NTFS for NT. NTFS is NT's native format, and NT supports the FAT and HPFS formats to provide migration paths from the other operating systems (although Microsoft didn't include support for HPFS in NT 4.0).

Microsoft's implementation goal for NTFS was to overcome the limitations of the other two NT-compatible file systems and provide the advanced features that an enterprise-level operating system requires. For example, NTFS supports file and directory-granular security, whereas FAT and HPFS have no security capability. In addition, NTFS's allocation scheme can efficiently address huge disks—FAT and HPFS both are limited to disk sizes that are challenged by today's commodity hardware. Finally, of the three file systems, NTFS is the only one that's Unicode-enabled for international environments and has built-in data reliability features to prevent file and file system corruption if the system fails.

In this article, I'll compare NTFS's capabilities with FAT's and explain the design of NTFS on-disk data structures. I'll wrap up with a preview of NTFS changes in NT 5.0 and mention several resources you can turn to if you end up with a corrupt NTFS drive. (For information about choosing a file system in NT 4.0, see Sean Daily, "NTFS vs. FAT," October 1996.)

NTFS Capabilities

In the late 1980s, Microsoft designed NTFS in parallel with the initial development of NT. After NTFS's basic infrastructure was in place and functional, Microsoft directed the NT programming team to use NTFS as the file system for its NT development systems. This arrangement created an effective feedback loop for the NTFS developers.

Because NTFS was to be a new file system from the ground up, its design could incorporate features that would overcome the limitations of existing PC file systems and anticipate the demands of NT's enterprise users down the road. The most obvious way to focus attention on NTFS was to make it scale with ever-expanding disks. All Windows file systems divide a disk partition into logical allocation units known as *clusters*. The FAT file system uses 16-bit entries to reference clusters, so at most FAT can address 2^{16} or 65,536 clusters. Clusters can vary in size according to the size of a disk, but large clusters result in *internal fragmentation*, or wasted space inside clusters. For example, if a file has only 250 bytes of data, it still requires an entire cluster of allocated disk space, which results in more than 15KB of wasted space on a disk with 16KB clusters.

With only 65,536 clusters of addressing capability, a FAT drive with 1KB clusters can cover a 65MB disk. Today's 4GB or larger disks require FAT clusters of 64KB—a size that typically yields large amounts of wasted space. In contrast, NTFS references clusters with 64-bit addresses. Thus, even with 512-byte clusters, NTFS can map disks up to sizes that we won't likely see even in the next few decades.

The developers of FAT and HPFS overlooked file system security, but NTFS uses the same security model as NT. Discretionary access control lists (DACLs) and system access control lists

(SACLs), which control who can perform operations on a file and what events will trigger logging of actions performed on a file, are stored in their native NT form within NTFS. This approach makes managing NTFS security a natural fit with general NT security administration.

The FAT file system uses the ASCII 8-bit character set in its file and directory name scheme. Employing the ASCII character set constrains FAT names to English (and some symbolic) characters. NT and NTFS both use the Unicode 16-bit character set for names. This attribute of NTFS lets NT users around the world manage their files using their native language.

FAT's model of data stored within a file is a one-unit approach. In contrast, NTFS lets several data streams exist within a file. The NTFS unnamed data stream is equivalent to the traditional FAT view of file data, but NTFS named streams can refer to alternative units of data. The named-stream model has not come into widespread use on NTFS (there isn't even a Win32 programming API that can access this capability), but Services for Macintosh (SFM) relies on the named-stream model to transparently let Macintosh clients use named streams on NTFS network drives as they do on their native Hierarchical File System (HFS) drives.

Finally, FAT has no provision for fault-tolerance. If a system crashes while you are creating or updating files and directories, FAT's on-disk structures can become inconsistent. This situation can result in the loss of the data being modified, as well as a general corruption of the drive and the loss of much of the disk's data. This risk is unnecessary and clearly unacceptable for NT's target market. NTFS has built-in transactional logging so that whenever a modification is about to take place, NTFS makes a note of the modification in a special log file. If the system crashes, NTFS can examine the log file and use it to restore the disk to a consistent state with minimal data loss.

NTFS Disk Management

Disk Administrator (windisk.exe) and the command-line Format utility are two tools you can use to create NTFS disk partitions and logical drives. Both tools let you specify exactly what size you want the clusters to be. However, if you choose not to specify an override, both tools will select default sizes based on the size of the disk. The tools make choices based on the amount of wasted space that can result from large clusters versus the amount of overhead that comes with managing small clusters. Table 1 summarizes the default cluster sizes.

TABLE 1: Default Cluster Sizes

Volume Size	Cluster Size
512MB or less	512 bytes
513MB-1024MB	(1GB)1KB
1025MB-2048MB	(2GB)2KB
2049MB or more	4KB

In addition to storing data belonging to your files and directories, NTFS uses several files to store data associated with disk management. The data stored within these files and any NTFS-related data stored in user files and directories is known as *metadata*. When you initialize an NTFS disk, NTFS creates 11 metadata files. Table 2 lists the metadata files and describes their purpose. These files are typically invisible to you when you browse an NTFS drive from the command prompt or from Explorer, but you can see them if you know what to look for. You can reveal information about a metadata file from the command prompt by typing

```
dir /ah <filename>
```

Screen 1 displays the results of using this technique for several metadata files.

TABLE 2: NTFS Metadata Files

NameMFT	Record	Description
$MFT	0	Master File Table—NTFS's command central
$MFTMIRR	1	Copy of the first 16 records of the MFT
$LOGFILE	2	Transactional logging file
$VOLUME	3	Contains volume serial number, creation time, and dirty flag
$ATTRDEF	4	Attribute definitions
.	5	Root directory of the disk
$BITMAP	6	Contains drive's cluster map (in-use vs. free)
$BOOT	7	Boot record of the drive
$BADCLUS	8	Lists bad clusters on the drive
$QUOTA	9	Contains user quota information—unused before NT 5.0 NTFS
$UPCASE	10	Maps lowercase characters to their uppercase version

I'll describe a few of the files in depth later, but I'll mention some files now. In addition to the logging that NTFS performs to prevent catastrophic loss of a disk, NTFS protects its on-disk data structures with signatures. When errors occur in reading NTFS data, NTFS identifies the clusters as bad clusters, remaps the data to another location of the disk, and updates the $BADCLUS file so that NTFS will avoid the faulty clusters in the future. If fault-tolerant disk drivers are present, they return information to NTFS that augments this hotfix capability to protect other data.

The $BITMAP file is a large array of bits in which each bit corresponds to a cluster on the drive. If the bit is off, the cluster is free; otherwise, the cluster is in use. NTFS maintains this file to track free clusters on the drive for allocating to new data.

The Master File Table

NTFS's command central is the Master File Table (MFT). The MFT is analogous to the FAT file system's file allocation table because the MFT maps all the files and directories on the drive, including NTFS's metadata files. The MFT is divided into discrete units known as records. In one or more MFT records, NTFS stores the metadata that describes a file or directory's characteristics (security settings and other attributes such as read-only or hidden) and its location on the disk. Surprisingly, the MFT is also a file that NTFS maps using records within the MFT. This structure lets the MFT grow and shrink, which makes the space NTFS uses for metadata efficiently track the amount of metadata that exists.

NTFS internally identifies files and directories using the position in the MFT of the record describing the start of their metadata. For example, the metadata files that Table 2 lists have pre-assigned starting (base) records in the MFT, which appear in the second column of Table 2. The records are usually 1KB in size on an NTFS 4.0-formatted drive, but they can be larger.

The $MFTMIRR file is another NTFS data-loss prevention measure. $MFTMIRR contains a copy of the first 16 records of the MFT, and NTFS stores the file in the middle of the disk (the MFT starts near the beginning of the disk). If NTFS has trouble reading the MFT, it refers to the

duplicate. An NTFS disk's boot record (512 bytes of data at the beginning of the disk) contains entries that locate the position of the MFT and the MFT mirror.

MFT-access performance plays a critical role in the overall performance of an NTFS drive, so NTFS takes a step to keep access fast. Because the MFT is a file on an NTFS drive, it can become fragmented as it grows and shrinks. This fragmentation results because NTFS cannot contiguously allocate the entire MFT in advance. When the MFT grows and other files use the clusters just beyond its end, the MFT must look elsewhere on the disk for available space, which causes discontinuous sequences of clusters in its file map.

The fastest file access occurs when a sequential disk operation can read entire chunks of a file, but a fragmented MFT means that NTFS may require multiple disk operations to read a record, which can lead to lower performance. In an effort to prevent MFT fragmentation, NTFS makes a region of clusters around the MFT off-limits to other files and directories. This area, the MFT-Zone, increases the chance that the MFT will find contiguous clusters or at least keep its data close to other MFT data when it needs to grow. When the disk space outside the MFT-Zone becomes scarce, NTFS relaxes its restriction and clusters from the zone can be allocated for other uses. Thus, the MFT runs a higher risk of becoming fragmented on disks that are near capacity. Unfortunately, NTFS does not let defragmentation tools defragment the MFT.

MFT Records

MFT records consist of a small header that contains basic information about the record, followed by one or more *attributes* that describe data or characteristics of the file or directory that corresponds to the record. **Figure 1** shows the basic structure of an MFT record. The header data includes sequence numbers that NTFS uses for integrity verification, a pointer to the first attribute in the record, a pointer to the first free byte in the record, and the MFT record number of the file's base MFT record if the record is not the first.

NTFS uses attributes to store all file and directory information. NT 4.0 NTFS has 14 attribute types, which Table 3 lists. On disk, attributes are divided into two logical components: a header and the data. The header stores the attribute's type, name, and flags, and it identifies the location of the attribute's data. NTFS uses a nifty performance-optimizing trick: It stores the attribute data within MFT records whenever possible, rather than allocating clusters elsewhere on the disk. When an attribute has its data stored in the MFT, the attribute is *resident*; otherwise, it's *nonresident*. A resident attribute is possible only when the attribute data fits within a record in addition to the record header, the attribute header, and other attribute headers. Thus, a hard upper-limit of 1KB (the common MFT record size) exists for the size of the data on most NT 4.0-formatted drives. If an attribute's data is resident, the attribute header points to the location of the data within the MFT record. By definition, the file name, standard information, and security attributes are always resident.

TABLE 3: NTFS Attribute Types

Attribute Type	Description
$VOLUME_VERSION	Volume version
$VOLUME_NAME	Disk's volume name
$VOLUME_INFORMATION	NTFS version and dirty flag
$FILE_NAME	File or directory name
$STANDARD_INFORMATION	File time stamps and hidden, system, and read-only flags

$SECURITY_DESCRIPTOR	Security information
$DATA	File data
$INDEX_ROOT	Directory content
$INDEX_ALLOCATION	Directory content
$BITMAP	Directory content mapping
$ATTRIBUTE_LIST	Describes nonresident attribute headers
$SYMBOLIC_LINK	Unused
$EA_INFORMATION	OS/2-compatibility extended attributes
$EA	OS/2-compatibility extended attributes

If NTFS must store an attribute's data outside the MFT, the attribute header, which is in an MFT record associated with the attribute's file or directory, contains information that locates the data on the disk. The data-mapping information is known as *run-information* because contiguous pieces of the data form one run made up of a starting cluster and length. The run-information, like most NTFS data structures, has a header that identifies which clusters of the attribute's data are mapped in the run-information. This approach is necessary because attributes with large amounts of data may have run-information split across multiple MFT records—each piece of the run-information covers different parts of the file. A run entry contains a *virtual cluster number* (VCN), which is a relative cluster offset within the attribute's data; a *logical cluster number* (LCN), which is the location on the disk where the data resides; and the number of contiguous clusters at that position on the disk.

Compressed files are a special case. NTFS supports compression on only the data stream of a file, and it applies the compression algorithm to blocks of 16 clusters. NTFS stores the compressed data on disk and requires clusters, but the saved space in a 16-cluster block is essentially stored in the compressed portion. The VCNs for these compressed clusters have nonexistent LCNs, which NTFS represents with an LCN of -1 in the compressed clusters' run entries.

If a file has too many attributes to fit within one MFT record, NTFS allocates additional records and stores an *attribute-list* attribute in the base record. The attribute list points at the location of attributes in the additional records and consists of an entry for each attribute.

Let's stop and look at an example that demonstrates some of these concepts. Mark.txt is a relatively large and fragmented file. Mark.txt requires a filename attribute, a data attribute, and a standard information attribute (it also has a security attribute, which I'll ig-nore to simplify the example). Thus, for mark.txt, the MFT will contain one or more records that contain the headers for these attributes, and in some cases the attribute data. In this example, which **Figure 2** depicts, the name and standard information are resident and stored within the first MFT record of the file. The data attribute is nonresident and is split into two attribute headers with runs in two records. The first run entry is two clusters of data at cluster 200 on the disk. The attribute-list attribute contains one entry that points at the data attribute header the second MFT record stores.

Directories

An NTFS directory is an *index* attribute. NTFS uses index attributes to collate file names. A directory entry contains the name of the file and a copy of the file's standard information attribute (time stamp information). This approach provides a performance boost for directory browsing because NTFS does not need to read the files' MFT records to print directory information.

When the entry data for a directory fits within an MFT record, one attribute type, the *index root*, describes the location of the entries in the record. As a directory grows, the entries required to describe it can overflow the file's MFT record. In these cases, NTFS allocates *index buffers* to store additional entries. The index allocation attribute header specifies the location of a buffer. On NT 4.0 NTFS, these buffers are 4KB in size, and the directory entries within them are variable length because they contain file names.

To make file lookups as efficient as possible, NTFS presorts the directory inside the index root and index allocation buffers. The sorting creates a tree structure: Buffers with lexicographically lower entries (e.g., 'a' is less than 'd') are beneath and to the left of higher-valued entries. Index allocation attribute headers contain run information just as nonresident data attributes do.

Figure 3 shows an example of a directory. As a simplification, the directory contains just a few entries, but the entries are split among the index root and two index allocation buffers. The red arrows show the direction that NTFS scans the entries when it performs a directory lookup. The black arrows show how the runs in the index allocation attribute reference the two allocation buffers.

NTFS Logging

As files and directories (including NTFS metadata files) change, NTFS writes records to the volume's log file. The CHKDSK program uses the log file to make NTFS on-disk data structures consistent and to minimize data loss in the face of a crash. The log file records come in two types: *redo* and *undo*. Redo records store information about a modification that must be redone if the system fails and the modified data is not on disk. For example, if a file deletion is in progress, the redo operation signals that the delete must be completed if a failure occurs and only some on-disk data structures have been updated.

NTFS uses an undo operation to roll back modifications that aren't complete when a crash occurs. If NTFS is appending data to a file and the system goes down between the time that NTFS extends the file's length and the time it writes the new data, the undo log file record tells NTFS to shorten the length of the file to its original size during a recovery.

The log file's size is based on the size of the disk and is usually between 2MB and 4MB. The log file will fill up unless NTFS ensures that the redo and undo records stored in the log file are not required for a recovery. Periodically, NTFS resets the data in the log file by sending all modified data to disk. This *checkpointing* process takes place about once every 5 seconds.

NT 5.0 NTFS

NTFS is gaining some exciting new features in its NT 5.0 incarnation. One new feature is built-in support for encryption, which will protect sensitive data on disks that might be viewable by programs such as NTFSDOS that ignore NTFS security settings (see "NTFSDOS Poses Little Security Risk," September 1996). NTFS doesn't perform the encryption; instead, an add-on piece (the Encrypting File System—EFS) is tied closely to the NTFS file system driver. EFS communicates with the system security authority and an EFS service, as well as NTFS. When a user wants a file encrypted, NTFS hands the data to EFS, which uses the user's security ID and a Data Encryption Standard (DES) encryption/decryption engine to manipulate the file's data. (The architecture is a little more complex than I've described, and worthy of a future column; the EFS architecture was not final at press time.)

File system quota add-ons for NT have been popular products, but they may lose their niche when Microsoft releases NT 5.0. NTFS has always hinted that it would natively support quotas (the $QUOTA metadata file has existed since NT 3.5), and NT 5.0 will finally implement them. NTFS

assigns quotas on a per-user basis, and the $QUOTA file in the new $Extend metadata directory stores quota specifications for the particular volume. NTFS honors this quota information, which can restrict a user to an upper limit on the amount of data on the volume in files that the user owns.

Other features include support for sparse files and reparse points. Sparse files can save space because NTFS allocates disk clusters for only parts of the sparse file that contain valid data—the portions of the file that don't contain valid data are assumed to contain 0s. Reparse points are essentially symbolic links, and can point at files or directories on the same or different disks.

Microsoft has stated that a "lite" version of Executive Software's Diskeeper defragmentation tool (see NT Internals: "Inside Windows NT Disk Defragmenting," May 1997) will be built into NT 5.0. This claim is a little misleading because Microsoft will just ship defragmentation tools (that use the NTFS deframentation APIs present since NT 4.0 and that work on NT 4.0 as well) in the box with NT 5.0. At press time, Microsoft had not made any changes to NTFS's defragmentation support for NT 5.0. I'll address other NT 5.0 features in the future article about EFS.

NTFS Resources

As you can see, NTFS is a relatively complex file system—the details about NTFS could fill a book. In fact, Helen Custer has written a good one: *Inside the Windows NT File System* (Microsoft Press, 1994). It takes you on a comprehensive tour of NTFS.

You can further explore NTFS at the NT Internals Web site (http://www.ntinternals.com). You'll find interesting information about NTFS (e.g., you can find out why NT won't place NTFS on floppy disks), and you'll find several useful NTFS data recovery tools (e.g., NTFSInfo, NTFSDOS, NTRecover).

As the changes coming in NT 5.0 show, NTFS is not a static file system. Each new version of NTFS is different enough that earlier versions won't recognize NTFS drives formatted with later versions. Microsoft has yet to write the final word on NTFS.

ABOUT THE AUTHOR

***Mark Russinovich** holds a Ph.D. in computer engineering from Carnegie Mellon University. He is coauthor of many popular NT utilities, including NTFSDOS, NTFilemon, NTRegmon, and NTRecover. You can reach him at http://www.ntinternals.com.*